CS783 Assignment-3

Amrit Singhal (150092)
Department of Computer Science
IIT Kanpur
Kanpur, India
amrits@iitk.ac.in

Aarsh Prakash Agarwal (150004)
Department of Electrical Engineering
IIT Kanpur
Kanpur, India
aarshp@iitk.ac.in

Problem Statement

The problem requires us to implement object detection in a two phase setting. In first phase, we are required to train a Resnet-18 model which given an image can detect the objects from the mentioned three classes- `aeroplane, bottle,` and `chair` . An additional class of `__background__` is added just to detect some default class, in case if the image does not belong to either of three classes. In second phase, i.e., during the test time, we run sliding windows of various sizes and aspect ratios on the test images. For each window, we calculate its allegiance to the classes. Finally we run the NMS on these windows to predict the classes with its bounding box.

ResNet Training

We need to train Resnet-18 model for four classes three classes- `aeroplane, bottle, chair`, and `__background__`. First step would be to process the input data.

Data preprocessing

For both the training and testing dataset, we extract the ground truth from the annotations given along with Jpeg images. For classes- `aeroplane, bottle,` and `chair`, the corresponding ground truths are cropped and are kept as training data for corresponding class. For `__ background __` class, for each image, a random box is cropped and it's IOU is checked with already present ground truths. If IOU is less than threshold value (0.5), the cropped image is taken as dataset for `__ background __` class. A small part of training data is kept for validation will training. Having the dataset ready, next task would be to train the required model. Resnet training is done in two different ways:

One layer training

In this training, the fully connected (fc) layer at the end of Resnet-18 pretrained model is changed to suit the **4** classes as output. Training is done with batch sizes of 128.

Two layer training

In this we were required to incorporate outputs from the previous layers of ResNet-18 model along with the output from the last layer. We have tried several versions/combinations and compared their accuracy before running final predictions on them. The combinations we have tried are as follows:

- **Two_layer_network_V1** In this version we took the Adaptive Average Pool of the output from the second layer of Resnet-18 model and added a fully connected layer with output size 4 for predicting the four classes at the end of it. We also, added a fully connected, again of output size 4 at last layer of the ResNet-18 model. Thus, we have two outputs, and we

calculated the loss for both the outputs independently and then added their losses to get the final loss of the network. This loss is appropriately back propagated to fine tune the weights of the model.

- **Two_layer_network_V2**: In this version instead of taking the output from the second layer, we took the Average Pool of the output from the first block of fourth layer of Resnet-18 model and added a fully connected layer of output size 4 to it. We also, added a fully connected, again of output size 4 at last layer of the ResNet-18 model. Like **V1**, we now have two outputs and we trained the model in a similar fashion.
- **Two_layer_network_V3**: In this version we first took the Average Pool of the output from the first block of fourth layer of Resnet-18 model and concatenated it with the output from the last layer of Resnet-18 and then added a fully connected layer of size 4 to it. Thus, we have two layers contributing independently to the output of the network. The loss is calculated and back propagated to fine tune the weights of the network.

Note that in all of the above models, we started with the pretrained weights of ResNet-18 model and only fine tuned its weights while training. For fine-tuning we have taken the help of Inkawhich.

Validation and Test Accuracy

We got the best test accuracy for the `Two_layer_network_V3` network. For both V1 and V2 we didn't get the great accuracy values therefore we didn't run prediction model on them. We didn't even completed the epochs for V2, as it's accuracy was so low. Only `Two_layer_network_V3` and `One_layer_network` are used for prediction as we didn't get time to run the other two. Time for training these networks was around 40- 50 minutes.

Model	Best Validation Accuracy	Test Accuracy
One_layer_network	**94.64%**	92.06%
Two_layer_network_V1	73.41%	74.24%
Two_layer_network_V2	66.22%	-
Two_layer_network_V3	91.97%	**92.29%**

Prediction

The objective of this section is to predict bound boxes for the classes `aeroplane, bottle,` and `chair`, given an image. These includes steps such as:

Sliding window

In this sliding windows of varying sizes and aspect ratio are run on the test image. If the window is classified as `aeroplane, bottle,` and `chair`, with confidence level greater than a threshold, its coordinates along with confidence level are given to NMS. For our experiments, we have tried with a total of 9 combinations of size, and aspect ratio with a fixed stride of 32. For this section, we have taken help from Rosebrock

NMS

For calculating NMS of predicted bounding boxes of a class on an image. We first sort the bounding boxes in the decreasing order of their confidence, then we start keeping bounding boxes from the top like a stack and remove all the bounding boxes with IOU greater than threshold (0.01 in our case) with the bounding box that we have kept. We keep doing this until the stack is empty, and all the bounding boxes we keep at last are the final ones that we return. For this section help was taken from Girshick.

Average Precision and mAP

For calculating the AP for a class. We first need to get the precision-recall values for that class. The first step would be to sort all the predicted bounding boxes in the decreasing order of confidence. Now, for each predicted bounding box (starting from the box with highest confidence), find whether it matches with any ground truth box of the corresponding image and class. For matching, just find the it's IOU with the boxes of it's class and corresponding image; if IOU comes out to be greater than 0.5, matching is successful. If the matching is successful, the prediction is regarded as TP(true positive), otherwise FP(false positive). For the i-th prediction count the number of TP and FP till that, and calculate precision as:

$$precision = \frac{TPs}{TPs + FPs}$$

You already know the total number of ground truths for that class. Calculate the recall till i-th iteration as:

$$recall = \frac{TPs}{Groundtruths}$$

Thus, you get precision-recall curve for a class. One can see that recall values are in increasing order. Starting with precision values corresponding to largest recall values, keep changing precision values to previous precision values unless you see an increase in them. This would give you the interpolated precision-recall curve. Now, get the precision values at 11 recall values ranging from 0.0 to 10.0 and get their average value to get the Average Precision for the class.

$$AP = \frac{1}{11} \sum_{\{0.0,0.1,\ldots,1.0\}} P_r$$

where P_r is precision value P at recall r.

Once you get the precision values for a class mean Average Precision (mAP) is calculated by average of AP for each class.

$$mAP = \frac{\sum_q AP_q}{Q}$$

For calculating mAP and AP Hui was referred.

AP and mAP values in our experiments are as follows:

Model	**AP** `aeroplane`	**AP** `bottle`	**AP** `chair`	**mAP**
One_layer_network	0.3322	0.1214	0.0569	0.1698
Two_layer_network_V1	1.0	0.1626	0.048	0.4035

Output Images

The output images for the One Layer detection model are: Note that **blue** indicates `aeroplane`, **green** indicates `bottle` while **red** boxes indicates class `chair`.

Note: We are giving images in report because it was very difficult to print them in the notebook. However, WE HAVE PRINTED THE NAME OF IMAGES WITH CORRECT PREDICTIONS IN THE NOTEBOOK WHILE TESTING.

One Layer Network

Correctly Predicted

Figure 1: Correctly Predicted bottle for 003431.jpg

Figure 2: Correctly Predicted aeroplane for 000067.jpg

Figure 3: Correctly Predicted bottle for 000762.jpg

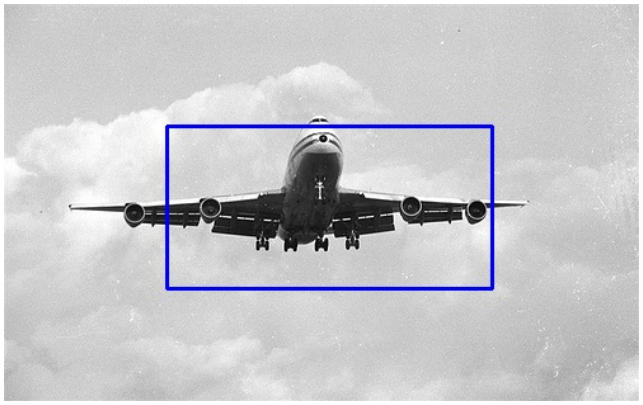

Figure 4: Correctly Predicted aeroplane for 001848.jpg

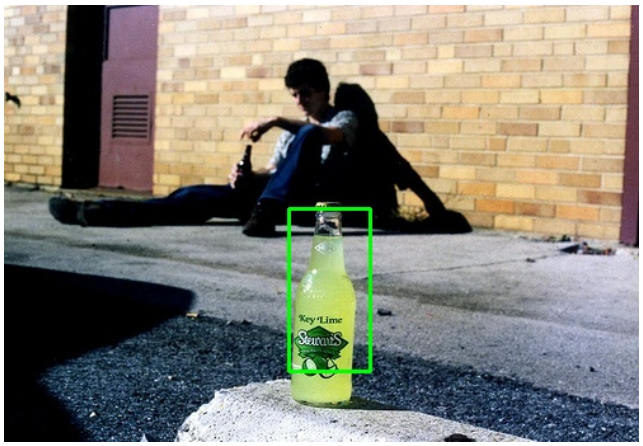

Figure 5: Correctly Predicted bottle for 000447.jpg

Incorrectly Predicted

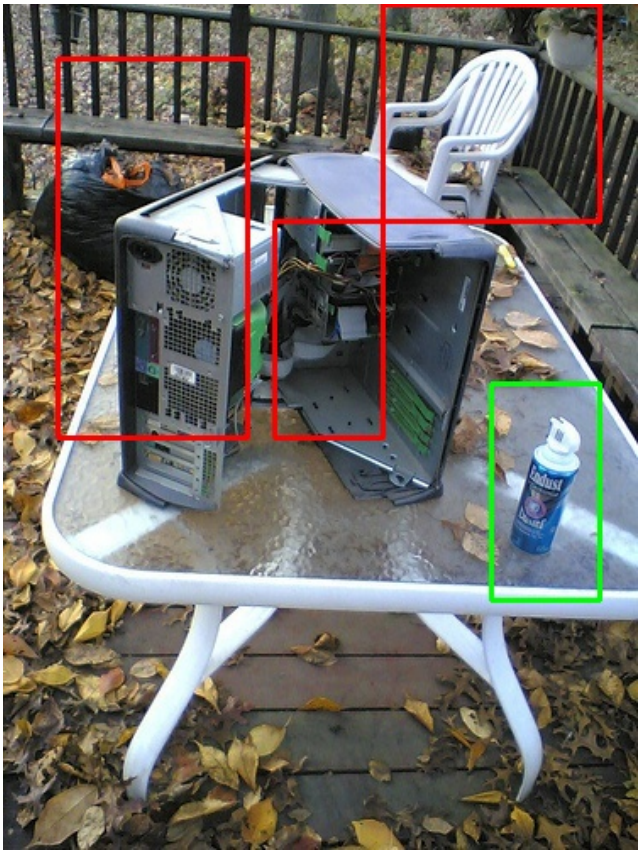

Figure 6: Incorrectly predicted CPU as chair for 002240.jpg

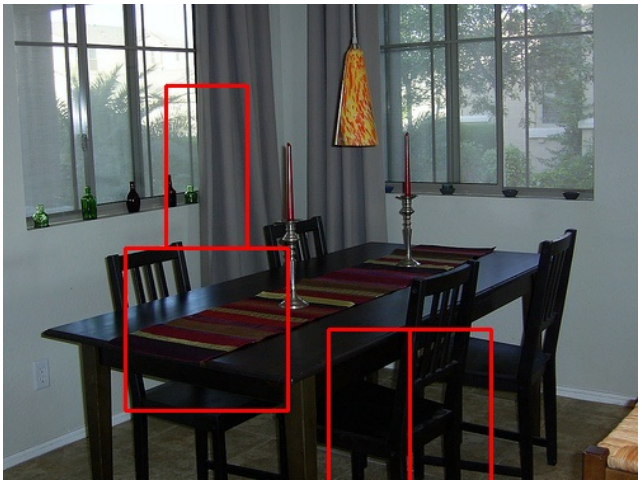

Figure 7: Wrongly predicted chairs along with some correct predictions for 002389.jpg

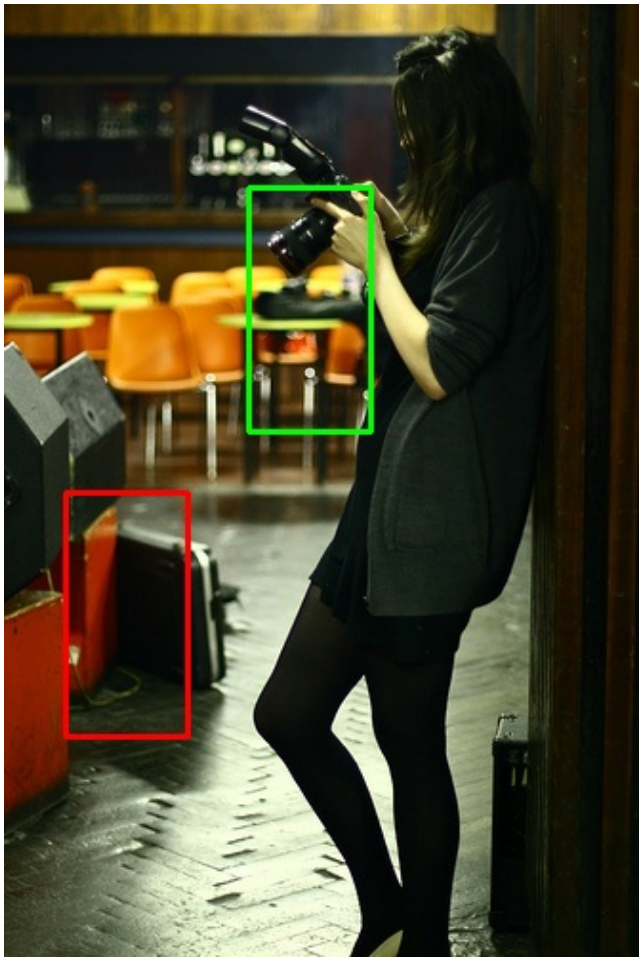

Figure 8: Complete random predictions for 002846.jpg

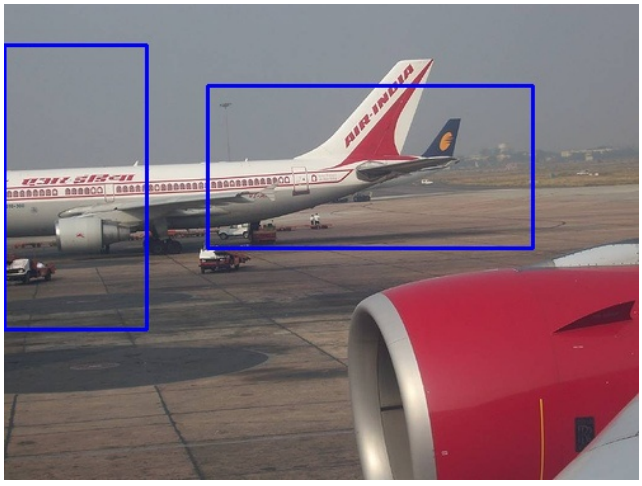

Figure 9: Wrongly predicted aeroplane for 002949.jpg

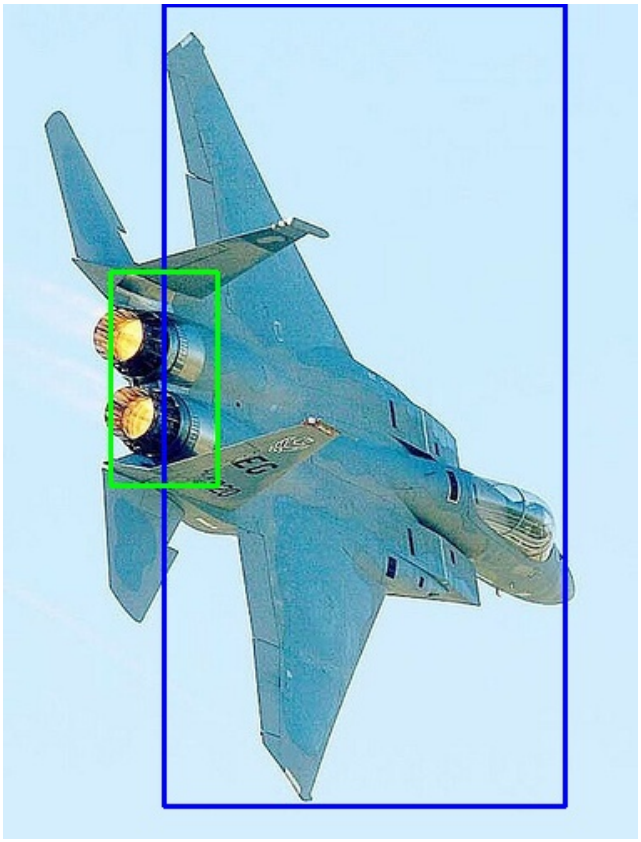

Figure 10: Wrongly Predicted bottle inside aeroplane for 002619.jpg

Two Layer Network

Correctly Predicted

Figure 11: Correctly Predicted bottle for 000369.jpg

Figure 12: Correctly predicted chairs for 000385.jpg

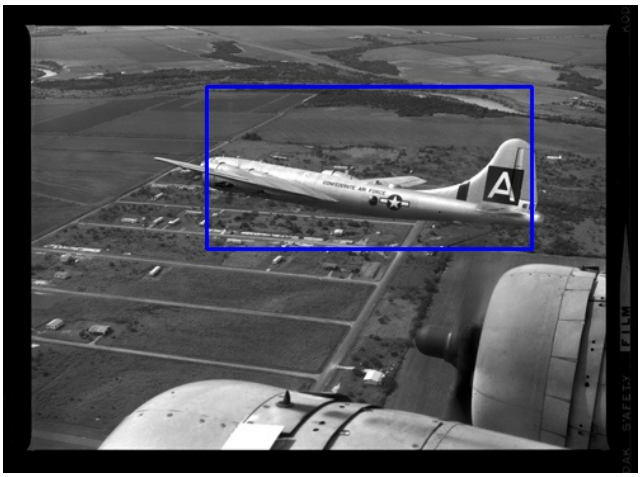

Figure 13: Correctly predicted aeroplane for 002198.jpg

Figure 14: Correctly Predicted aeroplane for 002246.jpg

Figure 15: Correctly Predicted aeroplane for 004314.jpg

Incorrect Predictions

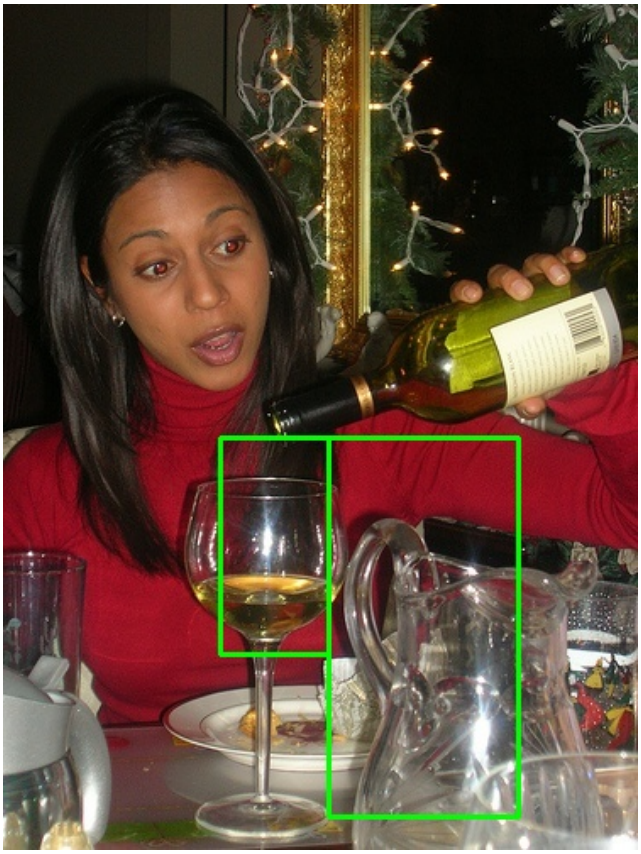

Figure 16: Wrongly Predicted bottle for 000151.jpg

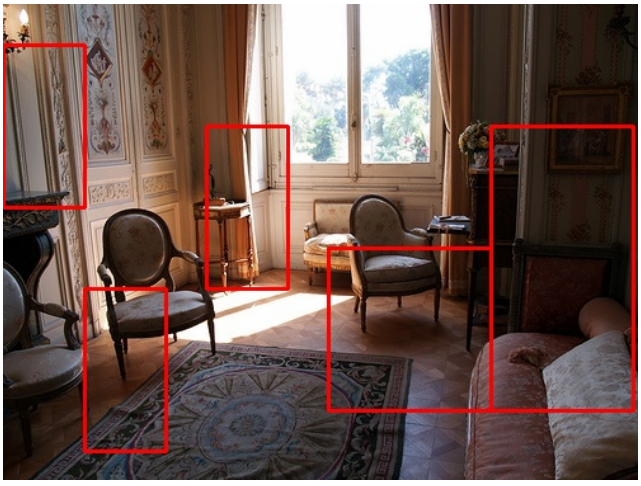

Figure 17: Wrongly Predicted some chairs for 000196.jpg

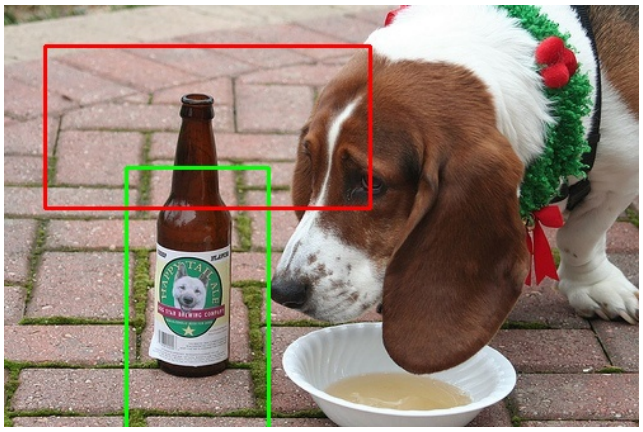

Figure 18: Incorrectly Predicted chair on bottle for 000611.jpg

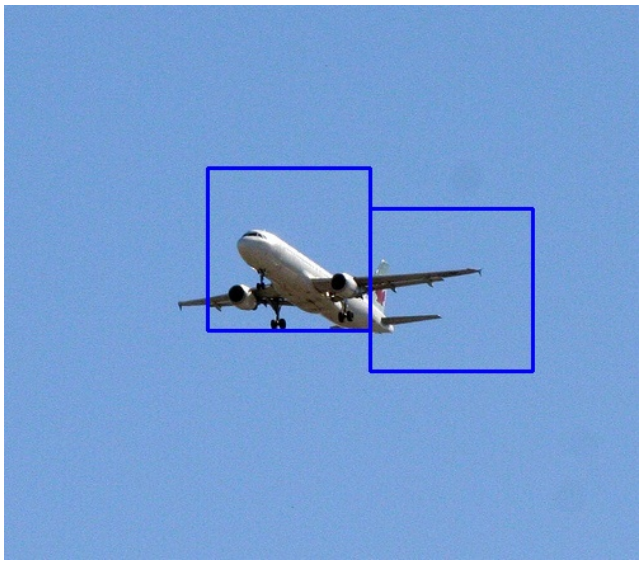

Figure 19: Incorrect prediction as two boxes predicted instead of one for 000665.jpg

Figure 20: Incorrectly predicted aeroplane as chair for 002971.jpg

References

Ross Girshick. Nms for fast rcnn. https://github.com/rbgirshick/fast-rcnn/blob/master/lib/utils/nms.py.

Jonathan Hui. map (mean average precision) for object detection. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.

Nathan Inkawhich. Finetuning torchvision models. https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html.

Adrian Rosebrock. Sliding windows for object detection with python and opencv. https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/.